AMARA Algorithm

Andrzej Szablewski

November 2, 2020

1 Introduction

The AMARA algorithm is an asymmetric (public key/private key) encryption method based on binary matrices developed in 2016 by Andrzej Szablewski, Radosław Peszkowski and Mateusz Janus under supervision of Prof. Tomasz Szemberg. The idea for creating such a method originated from the research on specific sets permutations [1].

2 Algorithm overview

Both public and private keys are considerably large binary matrices of a size n x n, which are inverse to each other. Data to encrypt or decrypt must be in the binary format. Cipher breaking difficulty lies in the matrix inversion complexity, which lower bound has been proven [2] to be:

$$O(n^2\ log(n)),$$

which is not an impressive complexity comparing to currently used methods. However, since AMARA encryption and decryption processes are based on *xor* operation between certain elements of the matrix, the algorithm is considerably quick.

3 Matrices generation

In order to create encryption-decryption binary matrices pair, which are inverse to each other, the creation process begins with an identity matrix of a certain size. Example of an identity matrix of size 3x3:

$$M_i = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3.1 Encryption matrix generation

Creating an encryption matrix (public key) is based on performing multiple *elementary operations* on a matrix. There are two *elementary operations* used in the algorithm.

1. Interchanging given two rows. For example interchanging rows 1 and 2:

$$M_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \longrightarrow \quad M_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2. Performing *xor* operation on every corresponding element of given two rows and storing the results on an indicated row. For example performing *xor* operations on rows 1 and 3 and storing result on row 1:

$$M_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \longrightarrow \quad M_2 = \begin{bmatrix} xor(1,0) & xor(0,0) & xor(0,1) \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \longrightarrow \quad M_2 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3.2 Decryption matrix generation

Creating a decryption matrix (private key) is as simple as performing exactly the same *elementary operations* that have been performed on the encryption matrix but in the reversed order.

4 Encryption

In order to encrypt a binary message using an encryption matrix of size n x n, the message must be cut into vectors, each of n elements. If the length of the message is not a multiple of n, the last vector must be complemented to the size of n with zeros. After encryption, all vectors are linked together to form the encrypted message - ciphert.

4.1 Vector encryption

Each element in the vector is corresponding to each row of matrix in the following way:

$$V = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \quad M_{EN} = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,n} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n,1} & m_{n,2} & \cdots & m_{n,n} \end{bmatrix}$$

Encrypted vector V_{EN} is initialized as follows:

$$V_{EN} = \begin{bmatrix} 0 & 0 & \cdots & 0 \end{bmatrix}$$

Algorithm iterates through the vector V and checks what is the value of each element v_i. If v_i value is 1, then it performs a *xor* operation on every corresponding element of the ith row of the matrix and vector V_{EN} and overwrites vector V_{EN} with the result.

5 Decryption

Decryption algorithm works exactly as the encryption algorithm but instead of using the encryption matrix M_{EN} it uses the decryption matrix M_{DE}.After decryption, all vectors are linked together to form the decrypted message.

5.1 Vector decryption

Each element in the vector is corresponding to each row of matrix in the following way:

$$V = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \quad M_{DE} = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,n} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n,1} & m_{n,2} & \cdots & m_{n,n} \end{bmatrix}$$

Decrypted vector V_{DE} is initialized as follows:

$$V_{DE} = \begin{bmatrix} 0 & 0 & \cdots & 0 \end{bmatrix}$$

Algorithm iterates through the vector V and checks what is the value of each element v_i. If v_i value is 1, then it performs a *xor* operation on every corresponding element of the ith row of the matrix and vector V_{DE} and overwrites vector V_{DE} with the result.

6 Vector encryption and decryption example

Lets set a pair of encryption-decryption matrices, which are inverse to each other:

$$M_{EN} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} M_{DE} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Let message vector be as follows:

$$V = \begin{bmatrix} v_1 = 1 \\ v_2 = 0 \\ v_3 = 1 \end{bmatrix}$$

6.1 Vector encryption example

$$V = \begin{bmatrix} v_1 = 1 \\ v_2 = 0 \\ v_3 = 1 \end{bmatrix} M_{EN} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

Lets initialize V_{EN} as follows:

$$V_{EN} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

1. Since $v_1 = 1$, algorithms performs *xor* operation on row 1 of the encryption matrix M_{EN} and vector V_{EN}:

   $$V_{EN} = \begin{bmatrix} xor(1,0) & xor(1,0) & xor(1,0) \end{bmatrix}$$

   which results in:

   $$V_{EN} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

2. Since $v_2 = 0$, algorithms moves to the next element of vector V.

   $$V_{EN} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

3. Since $v_3 = 1$, algorithms performs *xor* operation on row 3 of the encryption matrix M_{EN} and vector V_{EN}:

   $$V_{EN} = \begin{bmatrix} xor(1,1) & xor(0,1) & xor(1,1) \end{bmatrix}$$

   which results in:

   $$V_{EN} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$$

After iterating through the whole vector V, it has been encrypted and is stored in vector V_{EN}

6.2 Vector decryption example

Lets decrypt the vector V_{EN}, which was previously encrypted.

$$V_{EN} = \begin{bmatrix} v_1 = 0 \\ v_2 = 1 \\ v_3 = 0 \end{bmatrix} M_{DE} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Lets initialize V_{DE} as follows:

$$V_{DE} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

1. Since $v_1 = 0$, algorithms moves to the next element of vector V_{EN}.

$$V_{DE} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

2. Since $v_2 = 1$, algorithms performs *xor* operation on row 2 of the encryption matrix M_{DE} and vector V_{DE}:

$$V_{EN} = \begin{bmatrix} xor(1,0) & xor(0,0) & xor(1,0) \end{bmatrix}$$

which results in:

$$V_{DE} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$$

3. Since $v_3 = 0$, and this is the last element of vector V_{EN}, algorithm ends.

$$V_{DE} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$$

After iterating through the whole vector V_{EN}, the decrypted message is stored in vector V_{DE}.

Elements of the decrypted vector V_{DE} and the original message vector V should be identical and in fact they are.

$$V_{DE} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \quad V = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

References

[1] Andrzej Szablewski; Radosław Peszkowski; Mateusz Janus; Jakub Mazur. “Od problemu żarówek do nowego systemu szyfrowania (Polish) [From the lightbulb problem to the new encryption system]”. 2016.

[2] Ran Raz. “On the Complexity of Matrix Product”. In: *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*. STOC ’02. Montreal, Quebec, Canada: Association for Computing Machinery, 2002, pp. 144–151. ISBN: 1581134959. DOI: 10.1145/509907.509932. URL: https://doi.org/10.1145/509907.509932.